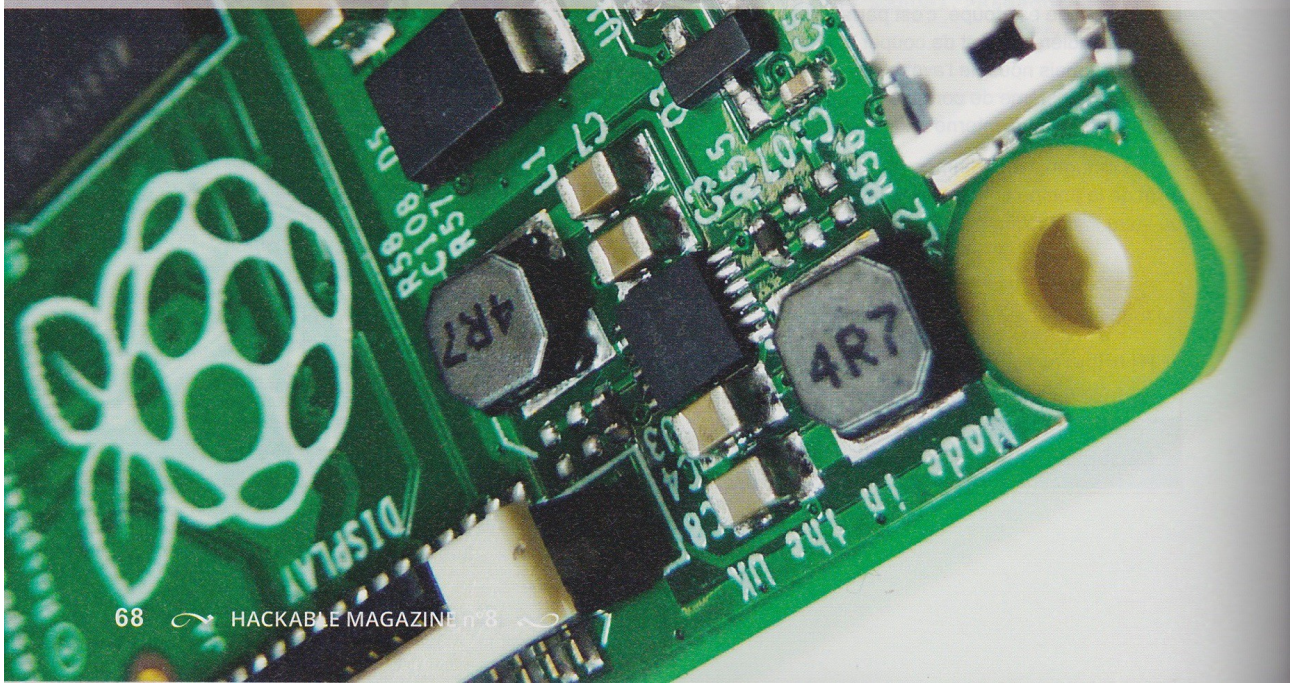
SUPERVISEZ VOTRE CONSOMMATION ÉLECTRIQUE SUR RASPBERRY PI

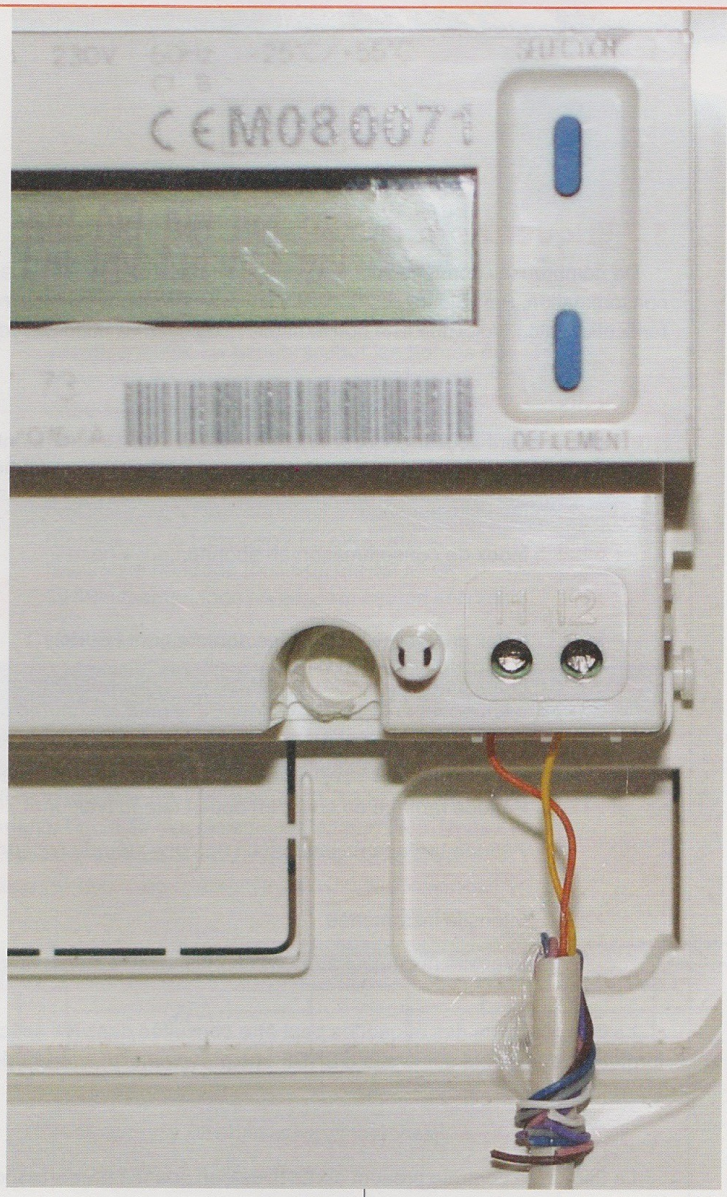
Sébastien Maccagnoni-Munch

Les compteurs de courant électroniques, installés dans les foyers français par ERDF, comportent une sortie appelée « téléinformation », qui permet de recevoir le détail des informations sur un autre appareil. Habituellement utilisée par des boîtiers appelés « gestionnaires d'énergie », cette sortie peut être connectée sur un ordinateur...



1. LA TÉLÉINFORMATION

En France, deux bornes présentes sur les compteurs électroniques (les compteurs blancs, les plus récents...) sont dédiées à la sortie « téléinformation », sur laquelle sont envoyés l'ensemble des informations concernant votre abonnement et votre consommation d'électricité. Ces bornes (nommées *I1* et *I2*) sont situées au bas à droite du compteur. Bien sûr, cette sortie n'existe pas sur les compteurs électromécaniques (à roue), qui n'embarquent pas d'électronique. Par ailleurs, elle est reprise sur le nouveau compteur « intelligent » Linky, actuellement en expérimentation, qui sera déployé à grande échelle dans les années qui viennent. Enfin, il faut que cette sortie ait été activée, ce qui n'est pas systématique ; quand ce n'est pas le cas, il faut faire une demande au fournisseur d'énergie afin qu'il planifie le déplacement d'un agent ERDF...



La sortie de téléinformation d'un compteur.

1.1 Format des données

Cette téléinformation est envoyée sous forme de signaux modulés à 50 kHz : la présence de modulation correspond à un 0 logique, l'absence de tension à un 1. Les informations remontées sont envoyées en continu, de manière cyclique, sous forme de caractères ASCII, à la vitesse de 1200 bauds. Chaque groupe d'informations (trame) commence par un caractère STX (*start of text*, code ASCII 2) et termine par un caractère ETX (*end of text*, code ASCII 3). Enfin, au sein de cette trame, chaque donnée est formatée de la manière suivante :

LF (*line feed*, ASCII 0x0A)	Étiquette (4 à 8 caractères)	Espace (ASCII 0x20)	Donnée (1 à 12 caractères)	Espace (ASCII 0x20)	Caractère de contrôle	CR (*carriage return*, ASCII 0x0D)

1.2 Étiquettes et données

De nombreuses données sont envoyées par votre compteur, liées à votre abonnement ou à votre consommation. Par ailleurs, les données envoyées diffèrent selon votre type d'abonnement (nombre de phases, tarification, etc.). Voici une liste non exhaustive de couples étiquette/données envoyés par le compteur :

Étiquette	Description	Unité	Nombre de caractères
OPTARIF	Tarif souscrit (valeur BASE, HC.., EJP. ou BBRx, x étant variable selon la configuration pour la sortie auxiliaire/déclenchement heure creuse)	/	4
ISOUSC	Intensité souscrite	A	2
BASE	Index de consommation en abonnement « base »	Wh	9
HCHC	Index des heures creuses en abonnement HP/HC	Wh	9
HCHP	Index des heures pleines en abonnement HP/HC	Wh	9
BBRHxJy	Index des six périodes tarifaires, pour un abonnement avec option « tempo » (x prend pour valeur « C » pour les heures creuses, « P » pour les pleines ; y prend pour valeur « B » pour les jours bleus, « W » pour les blancs et « R » pour les rouges)	Wh	9
PTEC	Période tarifaire en cours en option tempo (valeur BLEU, BLAN ou ROUG)	/	4
DEMAIN	Période tarifaire du lendemain en option tempo (valeur BLEU, BLAN ou ROUG)	/	4
IINST	Intensité instantanée	A	3
IMAX	Intensité maximale appelée, depuis la mise en service	A	3

La liste complète des étiquettes peut être consultée dans les spécifications détaillées de la sortie de téléinformation, disponible sur le site d'ERDF : **http://www.erdf.fr/sites/default/files/ERDF-NOI-CPT_02E.pdf**.

1.3 Caractère de contrôle

Le caractère contrôle présent après la valeur, quant à lui, correspond au calcul suivant :

- on fait la somme des valeurs ASCII de tous les caractères allant du début de l'étiquette à la fin de la donnée, espace incluse ;
- on ne conserve que les 6 bits de poids faible (ce qui correspond à un ET logique entre la somme ci-dessus et la valeur 3F en hexadécimal) ;
- on ajoute 20 en hexadécimal.

Le résultat sera toujours un caractère compris entre 20 (le caractère) et 5F (le caractère de soulignement ou *underscore*).

Par exemple, pour obtenir le caractère de contrôle pour l'étiquette « ISOUSC » avec la valeur 45, on peut exécuter la commande suivante :

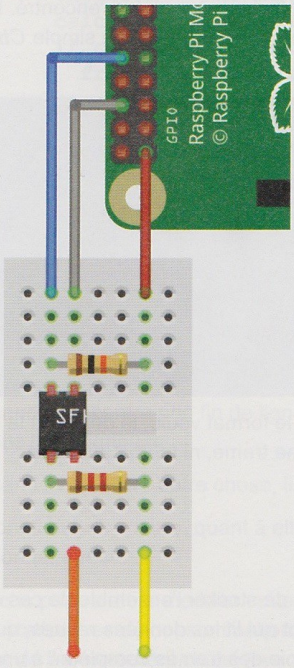
```
$ python -c "print chr((sum(bytearray('ISOUSC 45')) & 0x3f) + 0x20)"
?
```

On remarque, un peu plus loin dans cet article, que c'est bien le caractère « ? » qui est envoyé par le compteur pour cette valeur.

2. CONNEXION AU RASPBERRY PI

Pour lire ces données, nous allons brancher cette sortie de téléinformation sur l'entrée série d'un Raspberry Pi (nous avons également testé et validé cela avec un Beaglebone Black). Mais pour cela, il faut transformer ce signal modulé en signal logique : le Raspberry Pi n'est pas capable d'interpréter cette modulation de lui-même.

Pour cela, on va utiliser un optocoupleur SFH620A, accompagné de deux résistances...



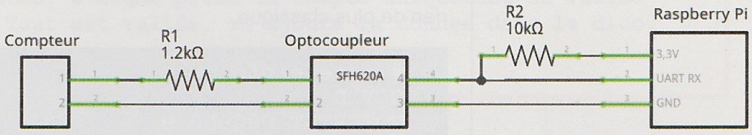
Bornes I1 et I2 du compteur

Le montage est très simple :

- d'un côté, les sorties de la téléinformation sont en « entrée » de l'optocoupleur, avec une résistance de 1,2 kOhm sur la patte 1 (il n'y a pas de polarité, on peut brancher les bornes de la téléinformation dans n'importe quel sens) - entre la sortie du compteur et le montage, on peut utiliser un câble à paires croisées de type câble téléphonique par exemple ;
- de l'autre côté, l'optocoupleur est directement branché sur le Raspberry Pi, avec une résistance de tirage (pull-up), qui est ici de 10 kOhm.

Ces trois composants peuvent être achetés ensemble chez certains vendeurs en ligne, sous le nom « kit téléinfo ».

Notons que 10 kOhm est une résistance trop élevée pour le pull-up sur un Beaglebone Black : cela fonctionne lorsque l'on descend à 3,3 kOhm, en mettant trois de ces résistances en parallèle. Nous n'avons pas testé d'autre matériel, à vous de vous assurer que cette résistance est bien dimensionnée...



Côté Raspberry Pi, on utilise trois broches :

- l'alimentation 3,3v (broche 1) ;
- la masse (broche 6) ;
- l'entrée du port série (UART RX, broche 10).

On ne branche pas la sortie du port série : ce canal de communication est unidirectionnel.

La configuration du port série nécessaire pour la téléinformation est la suivante :

- 1200 bits par seconde ;
- caractères sur 7 bits ;
- un bit de parité, paire ;
- un bit d'arrêt, 1 logique.

Ceci peut être résumé par la notation « 1200 7E1 ». On utilise donc 10 bits pour transmettre un caractère (7 bits de données, un bit de départ, un bit de parité et un bit d'arrêt) ; cela donne 120 caractères par seconde.

2.1 Lecture

Commençons par créer un petit script très simple en Python pour lire les données reçues sur le port série...

```
#!/usr/bin/env python

import serial, sys

s = serial.Serial(port='/dev/ttyAMA0', baudrate=1200, bytesize=serial.SEVENBITS,
                  parity=serial.PARITY_EVEN, stopbits=serial.STOPBITS_ONE)
while True:
  data = s.read(1)
  if data == '\2':   sys.stdout.write('\n===== DEBUT =====')
  elif data == '\3': sys.stdout.write('\n===== FIN =====')
  else:              sys.stdout.write(data)
```

On initialise d'abord le port série avec les paramètres adéquats, puis on lance une boucle infinie dans laquelle on lit les données caractère par caractère. Si on rencontre un STX, on indique qu'il s'agit d'un début de trame ; si on rencontre un ETX, on indique qu'il s'agit d'une fin de trame ; sinon on affiche le caractère rencontré. Pour sortir de la boucle, on ne prévoit rien : le programme s'arrêtera avec un simple *Ctrl-C*, rien de plus classique.

```
# ./teleinfo.py
C Y D
MOTDETAT 000000 B
===== FIN =====
===== DEBUT =====
ADCO 020828428827 J
OPTARIF BBR( S
ISOUSC 45 ?
BBRHCJB 026426186 @
BBRHPJB 041644955 P
^C
```

On constate qu'on reçoit bien les données dans le format voulu, le début de la réception n'étant pas nécessairement au début d'une trame, ni même au début d'une « ligne ».

2.2 Extraction

Étant donné qu'on travaille en Python, l'idéal est de stocker l'ensemble de ces valeurs dans un dictionnaire ! Créons alors un script qui lit les données reçues, qui valide les lignes, qui stocke les données et qui envoie des trames complètes à une fonction tierce...

```
#!/usr/bin/env python

import datetime
import serial

def got_frame(f):
  print f

if __name__ == '__main__':
  s = serial.Serial(port='/dev/ttyAMA0', baudrate=1200, bytesize=serial.SEVENBITS,
                    parity=serial.PARITY_EVEN, stopbits=serial.STOPBITS_ONE)
  char = None
  while char != '\x03': char = s.read(1) # Attendre une fin de trame
  while True:
    char = s.read(1)
    if char == '\x03': # Fin de trame, on envoie la trame en traitement
      got_frame(frame)
    elif char == '\x02': # Debut de trame, on reinitialise la trame
      frame = {}
    elif char == '\n': # Debut de ligne, on reinitialise la ligne
      line = ''
    elif char == '\r': # Fin de ligne, on traite les donnees
      try: tag, data, checksum = line.split()
      except ValueError: continue # Ligne invalide, on laisse tomber
      checksum = ord(checksum)
      calculated = (sum(bytearray(tag+' '+data)) & 0x3f) + 0x20
      if checksum == calculated: # Ligne prise en compte uniquement si valide
        frame[tag] = data # Tout est valide, on ajoute la donnee dans le dico
    else:
      line = line + char
```

Ce script reçoit les données du port série, caractère par caractère :

- quand il rencontre un début de trame, il crée un nouveau dictionnaire avec sa date de début ;
- quand il rencontre une fin de trame, il exécute la fonction `got_frame` avec le dictionnaire comme argument ;
- quand il rencontre un début de ligne, il crée une nouvelle ligne vide ;
- quand il rencontre une fin de ligne, il valide la ligne courante puis ajoute les données dans le dictionnaire ;
- quand il rencontre autre chose, il complète la ligne courante.

La fonction `got_frame`, quant à elle, ne fait pour l'instant qu'une impression à l'écran du dictionnaire reçu...

Chacun pourra alors modifier cette fonction afin d'effectuer les opérations souhaitées : suivi et statistiques de consommation, puissance nécessaire selon la journée, adéquation (ou non) de l'abonnement choisi (avec comparatif à d'autres tarifs), envoi des données à une autre machine...

3. FAIRE UN GRAPHIQUE

Pour avoir un début de résultat intéressant, on va générer un fichier d'historique des intensités instantanées, exploitable par **gnuplot**. Pour cela, la fonction **got_frame** va être remplacée par le code ci-dessous :

```
def got_frame(f):
  date = ('{d.year:04d}-{d.month:02d}-{d.day:02d}-{d.hour:02d}:'
          '{d.minute:02d}:{d.second:02d}'.format(d=f['date']))
  with open('intensite.data', 'a') as outfile:
    outfile.write('{} {} {}\n'.format(date, int(f['IINST']),
int(f['ISOUSC'])))
```

On obtient alors un fichier dont le contenu ressemble aux lignes suivantes :

```
2015-05-21-07:12:46 17 45
2015-05-21-07:12:48 16 45
2015-05-21-07:12:51 16 45
2015-05-21-07:12:53 14 45
2015-05-21-07:12:56 12 45
```

Ce fichier peut alors être transféré à tout moment vers un autre ordinateur, disons un poste de travail sous Linux. Pour exploiter ce fichier, on va alors créer sur ce poste de travail un fichier de définition de graphe pour **gnuplot**, que nous appellerons **intensite.plot** :

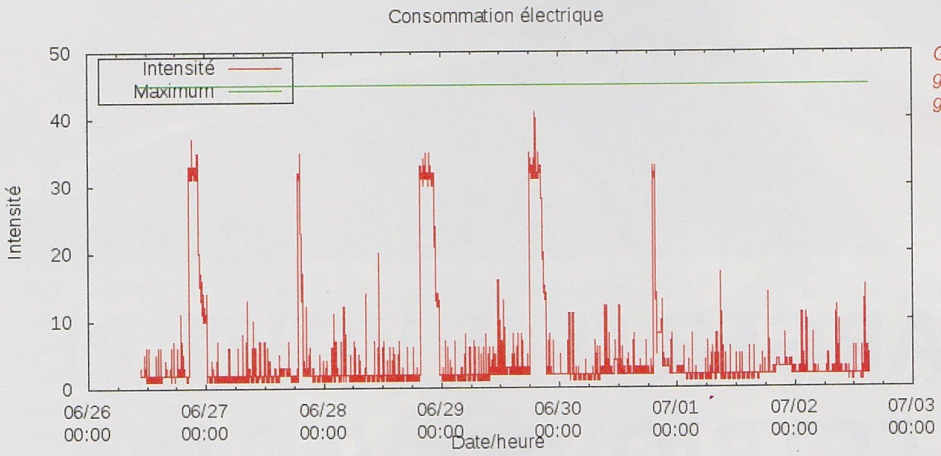
```
set terminal png size 800,400
set xdata time
set timefmt "%Y-%m-%d-%H:%M:%S"
set output "intensite.png"
set title "Consommation électrique"
set xlabel "Date/heure"
set ylabel "Intensité"
set key left box
set yrange [0:50]
plot "intensite.data" using 1:2 title "Intensité" with line,
"intensite.data" using 1:3 title "Maximum" with line
```

Il n'y a alors plus qu'à transférer le fichier **intensite.data** sur le poste de travail en question et à exécuter **gnuplot** :

```
$ gnuplot intensite.plot
```

... et l'image **intensite.png** (ci-contre) sera générée.

Bien sûr, ceci n'est qu'une approche très simplifiée et limitée de l'exploitation de ces données : idéalement, on aurait un ensemble de fonctions bien plus complexe, qui pourrait générer des graphiques à la volée, basés sur les différents indicateurs remontés, notamment les index de consommation (**BASE**, **HCHC**, **HCHP** ou d'autres, selon l'abonnement en cours). L'utilisation que l'on en fera dépend simplement de notre objectif...



Graphique généré par gnuplot.

3.1 Daemon

Dans la mesure où ce programme tourne en continu, il peut être intéressant d'en faire un *daemon* : un processus qui se détache du terminal courant et qui tourne en tâche de fond. Pour cela, de nombreux bouts de code et modules Python existent sur Internet. On choisira ici d'aller au plus simple, en ajoutant quelques lignes (très simples et sans gestion d'erreur) après le test portant sur `__name__` et avant l'initialisation de l'interface série :

```
[...]
if __name__ == '__main__':
  import os
  pid = os.fork()
  if pid == 0:
    pid = os.fork()
    if pid != 0:
      os._exit(0)
  else:
    os._exit(0)
  os.close(0)
  os.close(1)
  os.close(2)
  s = serial.Serial(port='/dev/ttyAMA0', [...]
```

À partir de ce moment-là, le programme rendra la main et tournera en tâche de fond dès qu'on l'exécutera.

4. OUTILS EXISTANTS

Ce n'est bien sûr pas la première fois que quelqu'un lit les données de téléinformation sur son ordinateur ! Différents logiciels ont déjà été faits pour cela, plus ou moins aboutis, plus ou moins flexibles. Avant de réinventer la roue, peut-être voudrez-vous vous intéresser à ces outils...

On pourra remarquer en particulier *teleinfuse*, qui permet d'accéder à ces données sous forme de pseudo-fichiers : **https://github.com/neomilium/teleinfuse**.

Le site personnel suivant est particulièrement fourni à ce sujet : **http://vesta.homelinux.free.fr/wiki/demodulateur_teleinformation_edf.html**. SMM